ISSN 0254-2757

QI QUAESTIONES INFORMATICAE

Volume 4 Number 3 October 1986

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R10	$ 7	£ 5
Institutions	R15	$14	£10

Circulation and Production

Mr C.S.M. Mueller
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg, 2001

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

LOW-COST ARTIFICIAL INTELLIGENCE RESEARCH TOOLS

Philip Machanick
Computer Science Department
University of the Witwatersrand, Johannesburg

There has been a proliferation of low-cost AI tools. Consideration is given as to how significant research may be conducted using these tools, in the light of experience of developing AI tools on large, expensive machines. As a case study, research in Knowledge Engineering-Based Learning (KEBL) is detailed, with mention of the tools used in the project. Results of the KEBL research are not yet available, but the style of the research is an indication of the potential of the tools described.

CR categories: I.2.5 Artificial Intelligence Programming Languages and Software, J.3 Computer Applications: Life and Medical Sciences

1. INTRODUCTION

As interest in Artificial Intelligence (AI) has spread, versions of languages and other AI tools have become available on relatively cheap computers. Such tools raise the possibility for AI research related to needs of relatively poor parts of the world.

The background to the development of cheap AI tools is considered, in with consideration of the lessons of traditional expensive AI tools. Brief consideration is given to how the experience of Turbo Pascal relates to the process by which AI tools can be expected to develop.

With the present state of available tools, consideration is given to whether significant research is feasible. The case for tools for practical applications is not an issue here; the number of potential tools for that market is also growing at an impressive rate [18]. Although there are many flaws in lower-cost tools, potential candidates for interesting developments are noted.

Research in Knowledge Engineering-Based Learning (KEBL) is used as an example of how relatively cheap tools can be a basis for AI-style program development. So far, the tools described — despite some major flaws — have proved adequate to the task. The problems detailed (and the occasional spectacular operating system crash) are indications that the technology is not as mature as one would like it to be, however.

In conclusion, experience with the tools used for KEBL — ExperLISP and ExperOPS5 — is summed up, with some consideration of how XLISP addresses some of the deficiencies of more conventional LISP implementations.

2. BACKGROUND

Most LISP books aimed at the low-cost computer market [10, 21] deal with primitive versions of the language, on the assumption that "big-machine" versions are not accessible to their readers.

Dialects of Common LISP [22] such as GC LISP (IBM PC — not too cheap) [7], ExperLISP (Apple Macintosh — moderately cheap) and XLISP (IBM PC and Macintosh — public-domain) [4] are challenging this assumption. Current exchange rates make the required computers expensive in South Africa, but US prices are declining.

A recent announcement in the personal computer field by Atari has a megabyte of RAM at under $1000 (albeit excluding a monitor) and is sure to put pressure on other manufacturers to drop prices, even if it does not turn out to be a major success. The significance of this development can be put into context by considering that the Interlisp environment was possible to develop because a large amount of memory was available by the standards of its time: 256K of 36 bit words of *virtual* storage [20]. If such a set of tools of acknowledged superiority could be developed with what now seems meagre resources, what will transpire now that so much computing power is so freely available?

It may be argued that Interlisp was put together by top-ranking researchers, and mass-market software will not achieve a similar level of sophistication. However, the lessons of the Interlisp experience are available for future developers of tools. Another positive factor is that a language such as LISP lends itself to incremental refinement [19]. Powerful tools — even in Interlisp — developed from humble beginings, with refinements developing as the need for them became obvious. Such an approach to development, while carrying the risk of uncontrolled proliferation by the undisciplined, is well-suited to mass-market tools, as the initial investment does not need to be high. An example of this is the development of tools around Turbo Pascal. Although Pascal is not seen as an AI language, Turbo has a significant feature in common with AI tools: an evironment which allows rapid recompilation after minor incremental changes.

The major difference between Turbo Pascal and AI tools comes in where large projects are being attempted: a good AI environment allows incremental recompilation, and has powerful tools to reduce the need for the programmer to keep track of bookkeeping details [20].

The extent to which the experience of Turbo Pascal can be applied to the development of cheap AI tools will be interesting to observe.

As an example of what can be achieved by such tools, a learning environment for Knowledge Engineering-Based Learning was implemented in ExperOPS5 — a version of OPS5 [6] implemented in ExperLISP. Despite the early stage of development of these tools, the implementation of OPS5 was viable for constructing a prototype of the required environment [15]. The final version was implemented directly in LISP to improve performance.

3. FEASIBILITY OF SIGNIFICANT RESEARCH

Most of the AI tools available in the past for small computers were primitive versions of languages like LISP [5] — or "expert system shells" with relatively few of the features of the systems used in major research projects [18].

Apple have announced a 1Mbyte version of the Macintosh — the MacPlus, which should be capable of supporting sophisticated tools. Many of the Macintosh's features — overlapping windows, mouse, powerful graphics — mimic features of sophisticated AI workstations [1, 20]. However, earlier versions were handicapped by having too little memory and disk drives which were too small and too slow. Since the Macintosh can be expanded to 4Mbyte without a major redesign, it has considerable potential for future development.

In the IBM range, the potential for expansion beyond the limitations of MS-DOS opened up by the PC AT are also being exploited: at least one full version of Smalltalk-80 [9] is available for this machine. Though a PC AT with all the required options is not cheap, it costs far less than a traditional AI machine — such as a Symbolics 3600 or LISP Machine.

Taking these points into account, there is not a lack of suitable hardware for the development of low-cost AI environments; suitable software is the missing ingredient. At one end of the spectrum, software is available, but expensive. The first version of Smalltalk-80 implemented for the PC-AT, at $1000, is not as expensive as many less exciting tools, but is not particularly cheap. Nor is Nexpert [8], one of the more interesting-looking tools for the Macintosh, with a launch price of $5 000. These prices should be seen in contrast with prices like $60 000 for KEE (Knowledge Engineering Environment) [13], which runs on a LISP Machine, and ExperLISP for the Macintosh at $495.

Although ExperLISP has many worthwhile features, it is at an early stage of development. It lacks sophisticated tools for supporting programming-in-the large, such as those found in Interlisp. However, it does support incremental compilation, and has a multi-window user interface, which facilitates development of complex programs in a modular fashion — functions which are closely related can be grouped together in a single file, and many files can be use to build up a large program. Use of the mouse to selectively compile pieces of text allows simple testing: by compiling small pieces out of a function, it is possible to see exactly what happens at each stage of execution. It also has a sophisticated virtual memory system, which makes good use of the Macintosh's resources. On the other hand, it lacks one of the most elementary debugging tools: a break loop [4]. In most LISP interpreters, it is possible to examine values of variables in the context where a run-time error (or break caused by a breakpoint) occured. This is particularly valuable in a modern version of the language, using lexical scoping.

On the IBM PC, GC LISP is an implementation of a "subset" of Common LISP. Although it has enough of a the flavour of the full language to be useful as a teaching aid, it is not as cheap as ExperLISP ($495 for an interpreted version; a similar amount extra for a compiler), and lacks some key features — such as lexical scoping. Furthermore, while ExperLISP runs on a 512K Macintosh, GC LISP needs at least 640K on an IBM PC to perform realistically. Despite these limitations, it has some useful features: GMACS (a version of the well-known EMACS editor), more sophisticated debugging facilities than ExperLISP's (although with some flaws) and a tutorial program (the San Marco Explorer) [7].

Turbo PROLOG, with a launch price of $99, has the potential to open up a completely new market. If it has the same impact on the AI community as Turbo Pascal had on programmers (effectively displacing BASIC as the PC "standard"), interesting possibilities are opened up. Although some features of PROLOG which are difficult to compile have been left out , and typing has been introduced, the negative effects will depend on a specific programmer's style. The viability of this environment for serious research is worth investigating.

4. CASE STUDY: *KEBL*

Knowledge Engineering-Based Learning grew out of several concerns: the need for education to move away from rote learning where acquisition of sophisticated skills is concerned, the need to develop technologies suitable for a third-world context and a desire to exploit the potential offered by the steady improvement in performance of computers in relation to price.

Medical education was identified as an area in which the first concern was taken seriously by many educationalists [3]. Specifically, diagnostic skills are not easy to teach, and many students only move from memorization of book knowledge to knowing how to apply this knowledge in a clinical situation two to three years after graduating [12]. Given that a shortage of medical skills is widespread in underdeveloped parts of the world, medical education is an obvious area to tackle in terms of the second concern. Use of computers as diagnostic aids was rejected as a first step because the number of computers needed would be much higher. The points mentioned above in relation to the development of AI tools for cheap computers was a consideration in exploring an AI-related approach; the experience of expert system builders in honing their skills during knoweldge acquisition was another.

ExperLISP and ExperOPS5 on the Macintosh were chosen as the best compromise between cost and performance available at the time the research was initiated. These tools offer an approximation to those found on large systems, at a fraction of the cost. The relationship of ExperLISP to other AI tools has been detailed above. ExperOPS5 was found to be an adequate approximation to the language as implemented elsewhere [6], and was a useful prototyping tool, but was not fast enough for the response time required, so the final implementation was in LISP.

Most studies of how diagnosis takes place focus on the development of hypotheses, leading to the acceptance of a specific hypothesis as the diagnosis. The approach is very similar to that of scientific method — pieces of evidence are gathered in a way directed by the currently active hypotheses. Various different processes are concurrently occurring, including hypothesis activation or deactivation, confirmation of active hypotheses and denial of alternative hypotheses [11]. An approach to teaching this style of learning — called problem-based learning — has been advocated [3], but with disappointing results [2].

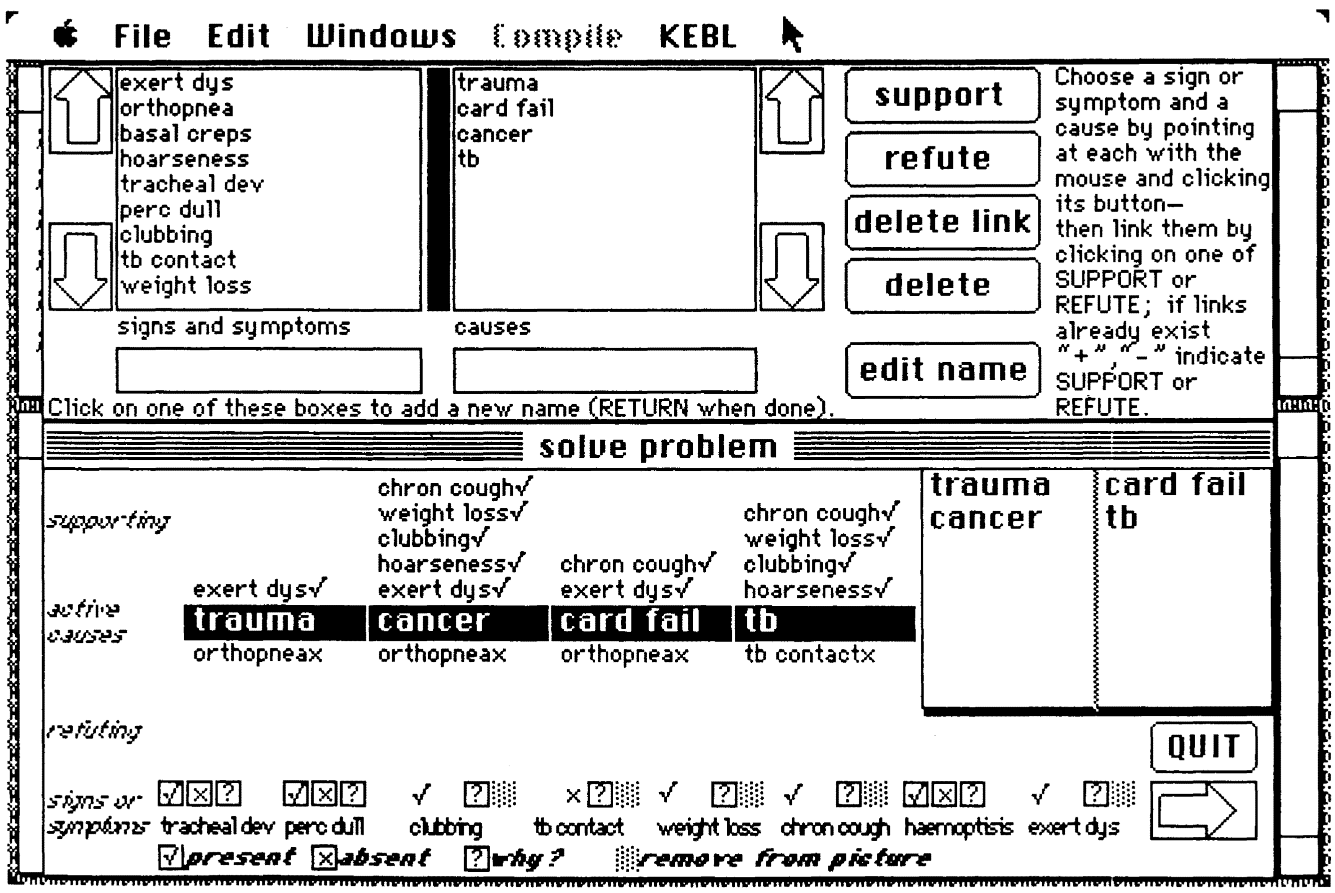
A major problem is that different individuals have different approaches to problem solving. For instance, given the same problem, one person could start by considering objective measurable medical facts, whereas another could consider the patient's social background — and both ought to arrive at the same diagnosis [17].

Doubts are being cast on the validity of labelling the problem-solving process as consisiting of evidence supporting or refuting hypotheses [16]. In order to avoid this debate, the more neutral terms "signs and symptoms" and "causes" were used instead.

Taking the experience of expert system builders into account, the construction of a simple expert system by medical students is a potential approach to developing their problem-solving ability. However, the construction of a full expert system requires significant skills which would take too much overhead to teach to non-computer scientists. The approach used in KEBL is to concentrate on discussion of a problem-solving approach in the context of a simplified approach to constructing and interpreting rules.

The rules take the form of `<name of sign or symptom> supports <name of cause>`, or `<name of sign or symptom> against <name of cause>`. Problem-solving is split into rule acquisition and tackling a specific problem. During knowledge acquisition, names of signs and symptoms and names of causes are added using the top half of the screen. A mouse pointing device is used to select the box in which a new name is to be typed. Once added to the system, names can be linked by selecting them and activating soft buttons on the screen to set up supporting or refuting links.

The specific problem phase, using the lower half of the screen, takes the form of activating or deactivating hypotheses, and activating (stating whether present or absent) or deactivating pieces of evidence. As an additional aid, it is possible to highlight all the hypotheses which relate to a specific piece of evidence. As with knowledge acquisition, the user interface relies heavily on using the mouse to activate "soft" buttons.



Choose a sign or symptom and a cause by pointing at each with the mouse and clicking its button— then link them by clicking on one of SUPPORT or REFUTE; if links already exist "+", "-" indicate SUPPORT or REFUTE.

Click on one of these boxes to add a new name (RETURN when done).

figure 1
The KEBL user interface

This situation gave rise to the following interaction between two simulated students (actually Computer Science lecturers) and a doctor:

doctor	what can you tell me?
first "student"	I think it's cancer
second "student"	I would say she has both TB and cancer
doctor	in fact, you couldn't tell the difference with this information; you would need to take an X-ray

The causes given in the rules appear in a table, and are activated or deactivated by pointing at them with the mouse and clicking the mouse button. Pieces of evidence are listed with buttons to add them ("√" for "present", "x" for "absent"), remove them or highlight the hypotheses relating to them. Once an hypothesis is activated, its name appears in the active area, where up to four hypotheses may appear. Supporting evidence appears above the name of the hypotheses; evidence against appears below (each tagged with a "√" or "x" to indicate its presence or absence).

An example of a KEBL screen appears in figure 1. In this instance, a practitioner entered the rules and simulated the patient while a Computer Science lecturer with no clinical knowledge attempted to gather information to form a diagnosis. Another Computer Science lecturer gave a second opinion. A more knowledgeable student would be expected to build up the rules as well.

By comparison with a conventional expert system, KEBL does relatively little. It does not weight signs and symptoms (evidence), form patterns explicitly, ask or answer questions or arrive at conclusions. The idea is that the user (and a tutor) would perform these functions. The student, by articulating the problem-solving process in a visible form, is in a position to discuss matters with a tutor (see figure 1). The formalization necessary to make the program run is also valuable as a focus for sorting out details. In many respects, the thought processes required are similar to those of the expert and knowledge engineer combined; the extra volume of work required to make the system function for another user in a general situation is missing.

In order to constrain the size of the domain, a well-known differential diagnosis is used as a starting point. More general diagnosis can be attempted at a later stage, but the approach is likely to fall down if the amount of knowledge acquisition needed before a specific problem can be dealt with becomes too high.

A preliminary study was carried out with first-year medical students (mainly because they were available at the time) [15]. Although the students had inadequate medical knowledge to carry out a thorough evaluation of the approach, some interesting points arose. For instance, a largely text-based user interface is a problem — it takes too much learning and the average student is a slow typist. Also, if the user interface constrains the approach too much, it inhibits exploration of alternative approaches. As a result, the user interface of figure 1 was designed. Not only does it require very little use of the keyboard, but all options are simultaneously available. The user is only constrained by the information currently in the system. In contrast to a conventional expert system, switching between knowledge acquisition and problem-solving is expected to happen at any time. It is possible, for instance, to modify rules in the middle of solving a problem.

An additional feature of KEBL is a record which is kept of the entire session. Statistical analysis of approaches to problem-solving, as well as playing a session back for evaluation and feedback, are possibilities which will be explored.

5. EXPERIENCE WITH TOOLS

ExperOPS5 may potentially be implemented more efficiently, but in the form used for the KEBL prototype, is not a practical proposition for a large project. It is, however, an excellent basis for learning about production systems, with a good range of debugging tools (single-step, display the conflict set, undo etc.), although better use of the Macintosh user interface could have been made.

ExperLISP, despite the positive points noted, is not without problems as well. It is very slow to load (at least a minute), and has some unexpected bugs, especially in relatively esoteric areas like menu manipulation and graphics programming. Furthermore, the current release does not support the Macintosh approach to programming its user interface through "resources", which means much which ought to be available through the operating system has to be explicitly programmed. Another problem is the operating system can easily be crashed by accessing invalid pointers. Clearly, from the point of view of providing safe access to such features as menu manipulation, a full class system (promised for the next release) is essential. Object-orientedness would also tie in naturally with the Macintosh interface, and would provide much-needed support for programming-in-the-large.

Even taking these negative points into account, the system is usable for fast prototyping and developing sophisticated programs. The key — as with any LISP system — is to develop suitable tools [14]. The current release of ExperLISP (version 1.5) is adequate for developing programs of up to 1 000 lines, or about 1 500 lines with a hard disk. Larger programs start exposing bugs in the compiler; in any case, developing programs much bigger than this would need tools for programming in the large (in the Interlisp style).

For an introduction to object-orientedness, XLISP is a worthwhile acquisition (especially as it is in the public domain). Although the syntax is sometimes clumsy, and large programs are likely to be unacceptably slow (it is not compiled), it is easy to generate some interesting examples, which make the lack of object-orientedness in other LISPs seem a major ommission. Furthermore, it is a good approximation to the spirit of Common LISP (it supports lexical scoping, unlike GC LISP), even if it is a relatively small subset of the language. On the Macintosh, the lack of a built-in editor can be overcome by using Switcher to run XLISP and a word processor more-or-less concurrently. The IBM PC version of XLISP can similarly be run using a resident editor such as Sidekick.

Since XLISP is continually being improved, perhaps it will rival some of the commercial products in time; $500 is a good lower bound on the price of sophisticated, usable (if flawed) AI tools.

REFERENCES

1. Alexander, Tom. The Next Revolution in Computer Programming, *Fortune* 29 October 1984 (65–70).
2. Babbot, David and Halter, William D. Clinical Problem-Solving Skills of Internists Trained in the Problem-Oriented System, *Journal of Medical Education* **58** 1983 (947–953).
3. Barrows, Howard S. and Tamblyn, Robyn M. An Evaluation of Problem-Based Learning in Small Groups Using a Simulated Patient, *Journal of Medical Education* **51**(1) January 1976 (52–54).
4. Betz, David. An XLISP Tutorial, *Byte* **10**(3) March 1985 (221–236).
5. Bortz, Jordan and Diamant, John. LISP for the IBM Personal Computer, *Byte* **9**(7) July 1984 (281–291).
6. Brownston, Lee, Farrell, Robert, Kant, Elaine and Martin, Nancy. *Programming Expert Systems in OPS5*, Addison-Wesley, Reading, Massachusetts, 1985.
7. D'Ambrosio, Bruce. Golden Common LISP, *Byte* **10**(13) December 1985 (317–321).
8. Dunn, Robert J. "Expandable Expertise for Everyday Users", *Computing SA* **5**(40) 14 Oct 1985 (14,15,18).
9. Goldberg, Adele and Robson, David. *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, Massachusetts, 1983.
10. Hasemer, Tony. *A Beginner's Guide to LISP*, Addison-Wesley, Wokingham, Berkshire, 1984.
11. Kassirer, Jerome P. and Gorry, G. Anthony. Clinical Problem Solving: A Behavioural Analysis, *Annals of Internal Medicine* **89**(8) August 1978 (245–255).
12. Leaper, D. J., Gill , P. W., Staniland, J. R., Horrocks, Jane C. and de Dombal, F. T. Clinical Diagnosis Process: An Analysis, *British Medical Journal* 3(9) 15 September 1973 (569–574).
13. Linden, Eugene. Intellicorp: The Selling of Artificial Intelligence, *High Technology* **5**(3) March 1985 (22–25, 82).
14. Machanick, Philip. Tools for Creating Tools: Programming in Artificial Intelligence, *Quæstiones Informaticæ* 3(3) August 1985 1985 (30–36).
15. Machanick, Philip. *A Study in Knowledge Engineering-Based Learning*, Computer Science Department Report CS–PM–86–008, University of the Witwatersrand, Johannesburg, 1986.
16. McGuire, Christine H. Medical Problem-Solving: A Critique of the Literature, *Journal of Medical Education* **60**(8) August 1985 (587–595).
17. Mitchell, Graham. Private Communication, 1986.
18. *PC Magazine*. **4**(8) 16 April 1985: special issue on expert systems.
19. Sandewall, Erik. Programming in an Interactive Environment: The 'LISP' Experience, *Computing Surveys* **10**(1) March 1978 (35–71).
20. Teitelman, Warren and Masinter, Larry. The Interlisp Programming Environment, Computer **14**(4) April 1981 (25–33).
21. Touretzky, David S. *LISP: A Gentle Introduction to Symbolic Computation*, Harper and Rowe, New York, 1984.
22. Winston, Patrick Henry and Horn, Berthold Klaus Paul. *LISP* (second edition), Addison-Wesley, Reading, Massachusetts, 1984.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0106
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM,* **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs and reprints

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.